Rancang Bangun Software Antivirus dengan menggunakan Metode Pendeteksian Heuristik

Deny Pradana, M. Komarudin, R. Arum S.P.

Jurusan Teknik Elektro Universitas Universitas Lampung
Jl. Prof. Sumantri Brojonegoro No.1 Bandar Lampung 35145
deny.n3t5ky@gmail.com

Abstrak

***Heuristik* memungkinkan *software antivirus* untuk mendeteksi *virus/worm* baru karena walaupun *virus/worm* sudah melakukan modifikasi terhadap *byte-byte* tertentu di *virus*, pattern atau pola *virus/worm* secara keseluruhan tidak akan berubah di mana hal ini akan memudahkan *antivirus* untuk mendeteksi *virus/worm* yang dilengkapi kemampuan *polymorph*. Dalam rancang bangun *software antivirus* ini digunakan bahasa pemograman *Visual Basic* 6.0 menggunakan metode *modified waterfall*. Dari hasil rancang bangun terlihat bahwa *software antivirus* ini mampu mendeteksi *virus/worm* baru yang belum terdapat pada *viruses signature database* seperti yang terlihat pada hasil pengujian, akurasi pendeteksian yang didapat tanpa menggunakan *heuristik* sebesar 61,53% dan dengan menggunakan *heuristik* sebesar 78.85%.**

Kata Kunci : ***Antivirus, virus, worm, malware, polymorph, heuristik***

Abstract

Heuristic enables antivirus software to detect new virus/worm because although the virus/worm has made changes in certain virus's bytes, most of the virus/worm pattern will not change and this thing will simplify antivirus to detect polymorph virus/worm. Visual Basic 6.0 and modified waterfall method was used in this antivirus design. From the design result, this antivirus software can detect new virus/worm that not included yet in viruses signature database that can be observed from the test result. Detection accuracy result without using heuristic was around 61,53% and with heuristic was around 78,85%.

Keywords : ***Antivirus, virus, worm, malware, polymorph, heuristic***

I. Pendahuluan

Perkembangan penyebaran *malware* di Indonesia pada awalnya lebih banyak didominasi oleh *worms* dan *virus* yang berasal dari luar negeri. Namun pada bulan Oktober 2005, dominasi ini mulai runtuh dengan menyebarnya *virus*/*worm* lokal yang hampir ada di setiap komputer di seluruh Indonesia, *virus* menyebar dengan sangat cepat dan sangat membuat risih bagi pengguna komputer, dengan demikian dibuatlah *software antivirus* sebagai salah satu solusi mencegah penyebaran.

Metode pencarian *virus* yang paling sering dipakai oleh *antivirus* yaitu metode pencocokan *CRC* (*Cyclic Redundancy Check*). Metode pencocokan *CRC* merupakan teknik yang semulanya digunakan untuk memeriksa kerusakan pada file. Metode ini yang sering digunakan oleh *antivirus* lokal untuk memeriksa *signature* dari virus, tetapi teknik ini tidak efisien apabila diterapkan pada *malware* yang sudah mengimplementasikan teknik *polymorph*. Pada kasus *virus* lokal sudah ditemukan penggunaan teknik *polymorph*. Baik itu secara sederhana maupun kompleks.

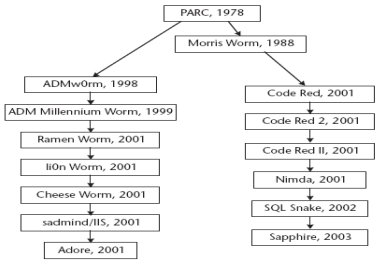
Penelitian ini menghasilkan *Software Antivirus* dengan Metode Pendeteksian *Heuristik* sehingga walaupun *virus*/*worm* sudah melakukan modifikasi terhadap *byte-byte* tertentu di virus, *pattern* atau pola *virus*/*worm* secara keseluruhan tidak akan berubah di mana hal ini akan memudahkan antivirus untuk mendeteksi *virus*/*worm* dengan kemampuan *polymorph.*

II. Tinjauan Pustaka

Virus/Worms komputer

Virus komputer pertama kalinya tercipta bersamaan dengan komputer. Pada tahun 1949 salah seorang pencipta komputer John von Newman, yang menciptakan *Electronic Discrete Variable Automatic Computer* (EDVAC), memaparkan suatu makalahnya yang berjudul "*Theory and Organization of Complicated Automata*". *Virus* dibuat oleh seseorang dengan tujuan yang bermacam-macam, tetapi umumnya para pembuat *virus* hanyalah ingin mengejar popularitas dan juga hanya demi kesenangan semata. Tetapi apabila seseorang membuat *virus* dengan tujuan merusak maka tentu saja akan

mengacaukan komputer yang ditularinya. Definisi umum *virus* komputer adalah program komputer yang biasanya berukuran kecil yang dapat menyebabkan gangguan atau kerusakan pada sistem komputer dan memiliki beberapa kemampuan dasar Terdapat beberapa jenis-jenis virus yaitu *virus* yang dibuat dengan *compiler*, *virus macro* dan *virus script/batch.* Sejarah *worms* mulai ada dan dikenal sejak awal internet mulai dipublikasikan. Di mana saat itu para ahli berusaha mengumpulkan informasi dari seluruh jaringan internet yang belum memiliki semacam mesin pencari (*search engine*). Untuk mengenal sejarah awal keberadaan *worms* ini, secara umum dapat dilihat pada gambar kronologis kemunculan *worms* berikut,



Gambar 1. Kronologis *worms*

Lima komponen yang umum dimiliki oleh *worm* adalah *recconnaissance, attack, communications, command* dan *intelligent*. Perbedaan *virus* dan *worms,* menurut Peter Szor (2005), *virus* adalah suatu program yang secara berulang (*recursively*) dan dengan tegas (*explicitly*) menggandakan suatu versi dirinya dengan mengikatkan dirinya (*attach itself*) pada program lain atau sistem operasi komputer sebagai kemungkinan untuk berevolusi dan penyebarannya memerlukan campur tangan pengguna. Menurut Jose Nazario (2004), *worms* adalah suatu agen penginfeksi otonom dan independen dalam bereplikasi, serta memiliki kemampuan dalam menginfeksi sistem *host* baru melalui fasilitas jaringan dan penyebarannya dengan atau tanpa campur tangan dari pengguna.

Pengertian *Antivirus*

Antivirus adalah sebuah jenis perangkat lunak yang digunakan untuk mengamankan, mendeteksi, dan menghapus *virus* komputer dari sistem komputer. Umumnya, perangkat lunak ini berjalan di latar belakang (*background*) dan melakukan pemindaian terhadap semua berkas yang diakses (dibuka, dimodifikasi, atau ketika disimpan).. *Antivirus* bisa dibagi menjadi tiga jenis yaitu *fix, antidot* dan *real-time antivirus*.

Cycle Redudancy Check (CRC)

Cycle Redudancy Check (CRC) adalah algoritma untuk memastikan integritas data dan mengecek kesalahan pada suatu data yang akan ditransmisikan atau disimpan. Data yang hendak ditransmisikan atau disimpan ke sebuah media penyimpanan rentan sekali mengalami kesalahan, seperti halnya *noise* yang terjadi selama proses transmisi atau memang ada kerusakan perangkat keras. Untuk memastikan integritas data yang hendak ditransmisikan atau disimpan, CRC dapat digunakan. CRC bekerja secara sederhana yaitu dengan menggunakan perhitungan matematika terhadap sebuah bilangan yang disebut sebagai *checksum*, yang dibuat berdasarkan total bit yang akan ditransmisikan atau yang hendak disimpan.

Dalam transmisi jaringan, khususnya dalam jaringan berbasis teknologi *ethernet*, *checksum* akan dihitung terhadap setiap *frame* yang hendak ditransmisikan dan ditambahkan ke dalam frame tersebut sebagai informasi dalam *header* atau *trailer*. Penerima *frame* tersebut akan menghitung kembali apakah *frame* yang ia terima benar-benar tanpa kerusakan, dengan membandingkan nilai *frame* yang dihitung dengan nilai *frame* yang terdapat dalam header *frame*. Jika dua nilai tersebut berbeda, *frame* tersebut telah berubah dan harus dikirimkan ulang. CRC didesain sedemikian rupa untuk memastikan integritas data terhadap degradasi yang bersifat acak dikarenakan *noise* atau sumber lainnya.

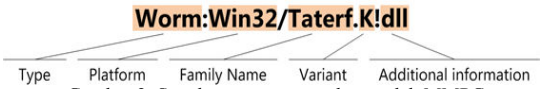
Antivirus Heuristic

Istilah *heuristic* berasal dari bahasa Yunani kuno yaitu *"heuriskein"*, yang dalam bahasa Indonesia memiliki kesamaan arti dengan kata "mengungkap". Di dalam penggunaannya, *heuristic* digunakan untuk menunjukkan teknik penyelesaian masalah melalui pendekatan pada pengetahuan dan kejadian-kejadian yang pernah dialami. *Heuristik* terbagi menjadi beberapa bagian pendekatan, yaitu *generate and test, hill climbing* dan *best first search*. Teknik *heuristic* pada *antivirus* berfungsi untuk menemukan *malicious code* yang tidak terdata pada *viruses signature database* dengan melihat karakteristik pada *file* seperti ukuran *file*, arsitektur atau *behavior* (tingkah laku) yang umumnya terdapat pada kebanyakan *malicious code*. *Heuristik* pada *antivirus* dibagi menjadi dua kategori yaitu *heuristik dinamis*, cara kerjanya dengan mengemulasikan *file* target pada suatu *virtual environment* (lingkungan virtual) dan meneliti tingkah laku *file* target pada *virtual environment* tersebut, kemudian *heuristik statis*, cara kerjanya dengan menggunakan *signature* tetapi tidak melihat spesifik pada *malicious software* tertentu melainkan dengan melihat *behaviour* (tingkah laku) dari suatu *file*.

Microsoft Malware Protection Center* (MMPC) *Naming Standars

Standar penamaan MMPC berasal dari skema penamaan *malware* oleh *Computer Antivirus Research*

Organization (CARO), aslinya diterbitkan pada tahun 1991 dan direvisi pada tahun 2002. Kebanyakan *vendor* keamanan komputer menggunakan penamaan *malware* berdasarkan skema CARO dengan sedikit perbedaan meskipun *family name* dan *variant* untuk *malware* yang sama dapat berbeda antara *vendor*. Standar penamaan yang digunakan oleh MMPC dapat berisi beberapa atau semua komponen-komponen berikut seperti yang tertera pada gambar berikut.



Gambar 2. Standar penamaan *malware* oleh MMPC

Diagram Aktivitas

Diagram aktivitas menggambarkan berbagai alir aktivitas dalam sistem, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. Diagram aktivitas juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. Diagram dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.

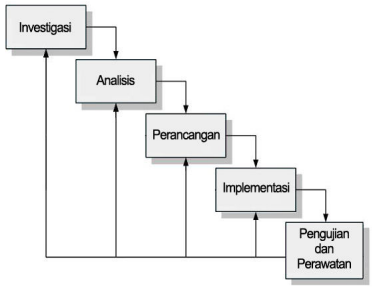
Pengenalan *Database*

Database merupakan kumpulan dari data yang saling berhubungan dengan yang lainnya, tersimpan di perangkat keras komputer dan digunakan perangkat lunak untuk memanipulasinya. Sistem *database* adalah suatu sistem informasi yang mengintegrasikan kumpulan dari data yang saling berhubungan satu dengan yang lainnya dan membuatnya tersedia untuk beberapa aplikasi yang bermacam-macam di dalam suatu organisasi. Dengan sistem *database* ini tiap-tiap orang atau bagian dapat memandang *database* dari beberapa sudut pandang yang berbeda. Semua terintegrasi dalam sebuah data yang umum.

III. Metode Penelitian

3.1 Tahapan Penelitian

Untuk pengembangan *software antivirus* ini digunakan metode *modified waterfall* dimana *modified waterfall* adalah sebuah metode pengembangan *software* yang bersifat sekuensial dan terdiri atas 6 tahap yang saling terkait dan mempengaruhi seperti terlihat pada gambar berikut.



Gambar 3. Model *Modified Waterfall.* (Sommerville, 2001)

1. Investigasi

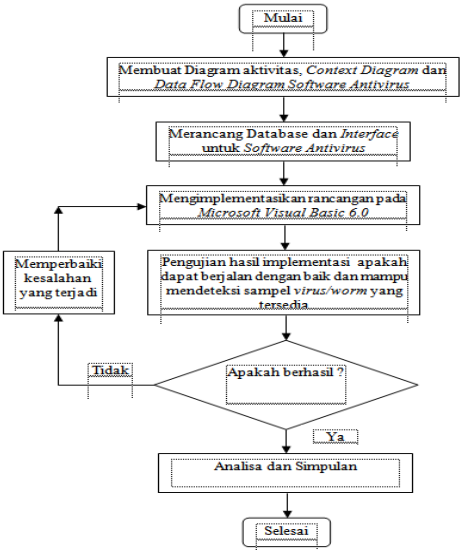
Untuk memulai pembuatan suatu sistem yang baru, harus dimulai dari awal dengan cara mengumpulkan informasi yang selengkap-lengkapnya. Pengumpulan data dilakukan dengan cara melakukan pengamatan terhadap beberapa *software antivirus* yang cukup populer di Indonesia, yaitu dengan mencoba secara langsung kemampuan beberapa *software antivirus* tersebut dalam mendeteksi *virus* dan *worm*. Dengan melihat cara kerja beberapa *software antivirus* tersebut, ditemukan beberapa kelemahan yaitu :

a. Terlalu bergantung pada *viruses signature database* untuk mendeteksi *virus*/*worm* baru.
b. Pengguna harus selalu melaporkan *virus*/*worm* baru yang belum terdeteksi kepada vendor *software antivirus* agar bisa ditambahkan ke dalam *viruses signature database* antivirus tersebut.
c. Penggunaan *resources* komputer yang cukup banyak dari beberapa *antivirus* sehingga membuat kecepatan komputer dalam memproses data berjalan dengan sangat lambat di mana hal ini justru menghambat kinerja dari *komputer* itu sendiri.
d. Tidak adanya fitur untuk memperbaiki sistem yang telah dirusak oleh *virus*/*worm*.

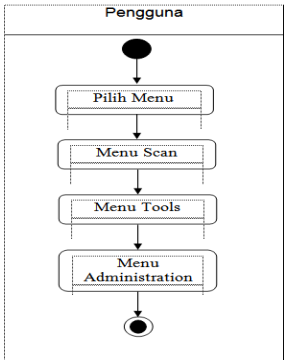
Software antivirus dengan metode pendeteksian *heuristik* yang akan dibuat sebagai sistem baru akan memperbaiki kelemahan-kelemahan tersebut, di mana pengguna nantinya tidak perlu terlalu sering memperbarui *main database* dari *antivirus*, bahkan pengguna bisa langsung menambahkan *virus*/*worm* baru yang belum terdeteksi oleh *antivirus* ke dalam *user-defined database* dan juga *antivirus* ini mampu berjalan dengan memanfaatkan *resource* computer yang sedikit serta dilengkapi fitur tambahan untuk memperbaiki sistem yang telah dirusak oleh *virus*/*worm*.

3.2 Analisis

Untuk sistem baru yang akan dibuat akan lebih dapat dipahami jika digambarkan dalam bentuk *flowchart* dan diagram aktivitas. Berikut ini adalah *flowchart* dan diagram aktivitas sistem baru yang akan dibangun.



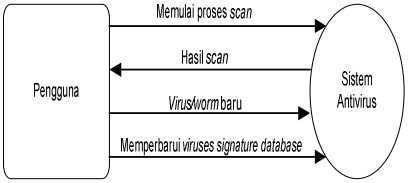
Gambar 4. *Flowchart* pembuatan *Software Antivirus*



Gambar 5. Diagram aktivitas menu utama

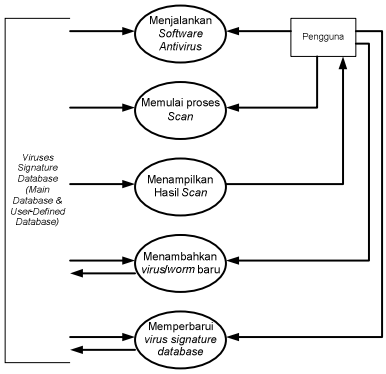
Data Flow Diagram

Langkah awal dalam perancangan sistem ini adalah pembuatan sistem *Context Diagram*. *Context Diagram* ini merupakan gambaran awal dari sistem pada *software antivirus* ini secara umum, yang menggambarkan sistem beserta hubungannya dengan lingkungan luar dan bagaimana sistem ini berinteraksi. Penjelasan sistem yang lebih rinci dapat dilihat pada *Data Flow Diagram*. Dari sini bisa didapatkan gambaran secara lebih jelas lagi tentang sistem yang akan dibangun.



Gambar 6. *Context Diagram*

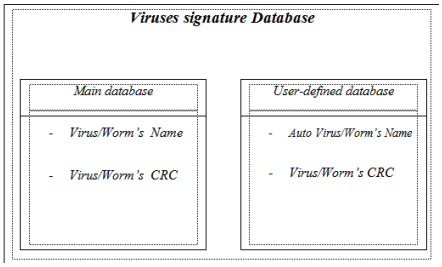
Dari *Context Diagram*, penggambaran proses-proses dalam sistem bisa lebih diperluas lagi dengan pembuatan *Data Flow Diagram*. Pembuatan *Data Flow Diagram* ini juga bertujuan untuk lebih memperjelas lagi hubungan antara sistem dengan kesatuan luarnya, karena dalam *Data Flow Diagram* kesatuan luar tersebut akan terhubung dengan proses-proses yang lebih relevan. Berikut ini adalah perancangan DFD dari sistem yang akan dibangun.



Gambar 7. *Data Flow Diagram Software Antivirus*

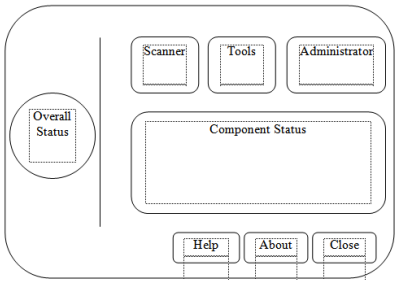
Perancangan *database*

Untuk perancangan *viruses signature database* yang terdiri atas *main database* dan *user-defined database*, maka stuktur tabel dan atribut yang akan dibuat adalah sebagai berikut.

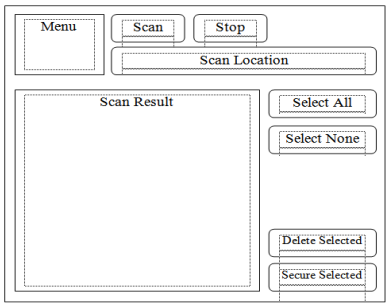


Gambar 8. Struktur tabel dan atribut dari *viruses signature database*

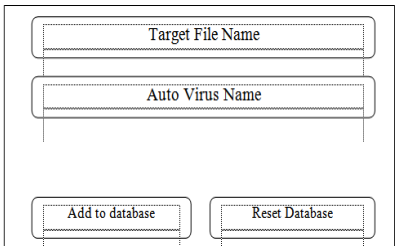
Perancangan *interface*



Gambar 9. Perancangan *interface* menu utama



Gambar 10. Perancangan *interface* menu *Scanner*

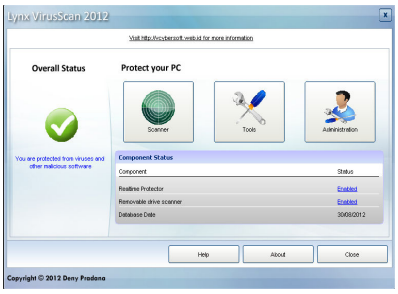


Gambar 11. Perancangan *interface* menu *User Defined Virus*

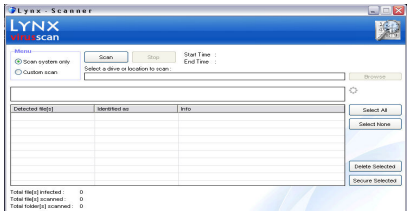
IV. Hasil Dan Pembahasan

Implementasi Antarmuka *Software*

Implementasi antarmuka *software* yaitu pembuatan tampilan *software* berdasarkan model yang telah dibuat. Tampilan yang dibuat berjumlah tujuh belas buah. Pengimplementasian antarmuka *software* ini berlangsung secara iteratif dan terus menerus dalam fase kontruksi sesuai dengan perkembangan aplikasi.



Gambar 12. Tampilan menu utama



Gambar 13. Tampilan menu *Scanner*

Gambar 14. Tampilan menu *User-defined Virus*

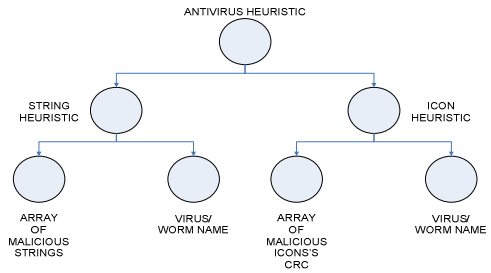
Implementasi basis data

Dari tahap perancangan basis data diimplementasikan menjadi basis data berupa *plaintext*. Berikut adalah format basis data yang digunakan pada sistem yang dibangun.

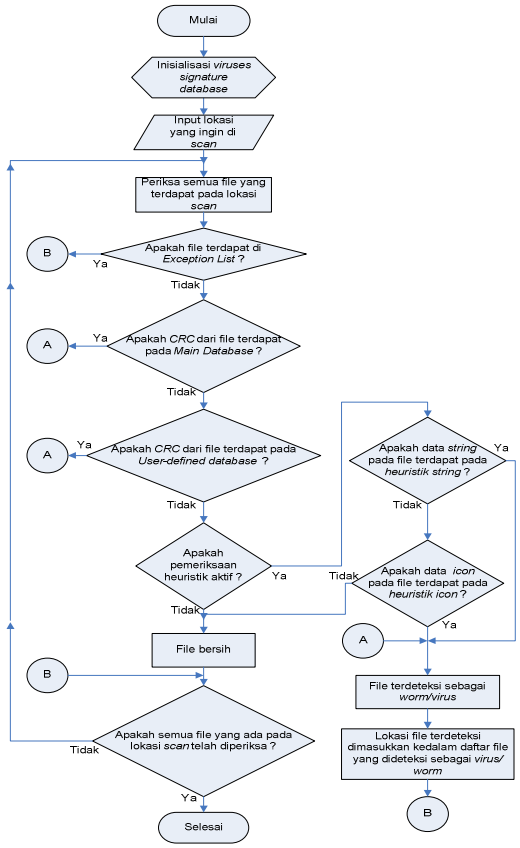
```
1  dbVer=17/10/2012
2  WORM:WIN32/BADASS=1085DD16
3  WORM:WIN32/LOVGATE.W=10C08823
```

Gambar 15. Format *Main database* (signatures.vdf)

Pada tahap ini, peneliti juga melakukan pengujian *software antivirus* yang telah dibuat. Tujuannya untuk mengetahui fungsionalitas *software antivirus* apakah telah sesuai dengan yang diinginkan. Kriteria pengujian dilakukan berdasarkan kriteria yang dibuat oleh sebuah lembaga independen bernama *AV-Comparatives*, yang bergerak dalam bidang *antivirus testing*. Tiga kriteria utama yang digunakan adalah *Real World Testing, File Detection Test* dan *Performance Test*. Metode pengujian yang digunakan adalah *black box testing* dan *white box testing*.



Gambar 16. Pohon *heuristik* yang digunakan pada *software antivirus* ini



Gambar 17. *Flowchart* proses *scanning* pada *software antivirus* yang dibuat

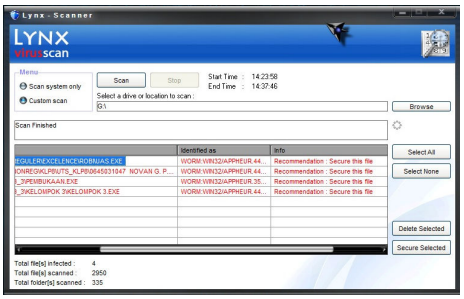
Metode *heuristik* yang digunakan oleh *software antivirus* ini adalah metode *heuristik statik* atau secara umum dinamakan metode *generate and test* dimana cara kerja metode *heuristik* ini adalah dengan membangkitkan/*generate* keseluruhan *string* yang ada pada *file* target terlebih dahulu, kemudian *software antivirus* akan membaca keseluruhan *string* dari file *target* dan mencocokkan dengan kumpulan *array* yang berisi *string-string* mencurigakan. Apabila ditemukan kesamaan antara salah satu *string* yang ada pada *file* target dengan salah satu *string* yang ada pada kumpullan *array*, *file* target tersebut akan dideteksi sebagai *virus/worm*. Tetapi apabila *heuristik string* ini tidak dapat menemukan *string-string* mencurigakan pada *file* target, maka pemeriksaan akan di lanjutkan ke pemeriksaan *heuristik icon* dimana *software antivirus* akan membaca *CRC icon* dari *file* target dan mencocokkan dengan kumpulan *array* yang berisi *CRC-CRC icon* yang mencurigakan. Apabila ditemukan kesamaan antara *CRC icon* yang digunakan *file* target dengan salah satu *CRC* yang ada pada kumpulan *array*, *file* target tersebut akan dideteksi sebagai *virus/worm*. Bila tidak ditemukan adanya *CRC icon* yang mencurigakan pada *file* target, maka *software antivirus* ini akan menganggap *file* target sebagai *file* yang bersih.

Tabel 1. Hasil pengujian tanpa menggunakan metode *heuristik*

Status Pemeriksaan *Heuristik*	Jumlah *file* yang terdeteksi	Jumlah sampel *virus/worm* yang ada	Akurasi Pendeteksian
Nonaktif	64	104	64/104 * 100% = 61.53 %

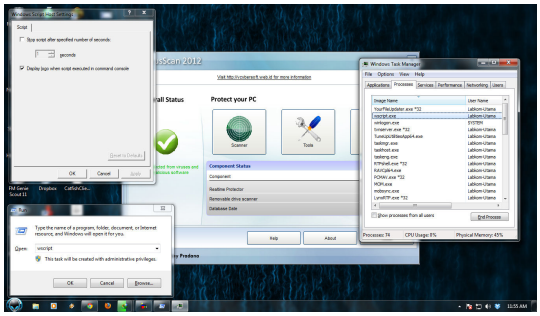
Tabel 2. Hasil pengujian dengan menggunakan metode *heuristik*

Status Pemeriksaan *Heuristik*	Jumlah *file* yang terdeteksi	Jumlah sampel *virus/worm* yang ada	Akurasi Pendeteksian
Aktif	82	104	82/104 * 100% = 78.85 %



Gambar 18. *Virus/worm* yang ditemukan pada sampel *flashdisk* dengan menggunakan metode pendeteksian *heuristik*

Setelah pengujian *heuristik*, dilakukan pengujian fitur *Real-Time Protector* yang terdapat di *software antivirus* pada sistem operasi *64-bit*. Pengujian dilakukan dengan menjalankan salah satu fitur pada sistem operasi *Windows* yang sering dimanfaatkan oleh *virus/worm* yaitu *Windows Scripting Host*. Setelah fitur tersebut berjalan, *Real-Time Protector* yang ada pada *software antivirus* tidak dapat menghentikan proses *Windows Scripting Host* tersebut seperti yang terlihat pada gambar 19.



Gambar 19. Fitur *Real-Time Protector* pada *software antivirus* tidak dapat menghentikan proses *Windows Scripting Host* pada *OS 64-bit*.

Selanjutnya adalah pengujian *Performance Test*. Pada pengujian ini diukur seberapa besar *resource* komputer yang digunakan oleh *software antivirus* sebelum melakukan proses *scanning* dan ketika melakukan proses *scanning*. Pengujian ini berfokus pada *memory usage* dan *CPU usage*.

i. Pengujian sebelum proses *scanning*

Pada pengujian sebelum proses *scanning, memory usage* dari *scanner* tercatat pada angka 7628 KB dan *CPU Usage 0%* seperti yang terlihat pada gambar 20.

Gambar 20. *Memory Usage* dan *CPU usage* sebelum proses *scanning* dilakukan

ii. Pengujian ketika proses *scanning*.

Pada pengujian ketika proses *scanning* berjalan, *memory usage* dari *scanner* tercatat pada angka 22608 KB dan *CPU usage* 25% seperti terlihat pada gambar 21.

Gambar 21. *Memory usage* dan *CPU usage* ketika proses *scanning* berjalan

Simpulan

Dari hasil dan pembahasan pada penelitian ini, dapat diambil beberapa simpulan sebagai berikut.

1. Akurasi pendeteksian *virus/worm* oleh *software antivirus* yang dibuat menggunakan metode *heuristik* lebih tinggi dibandingkan dengan tanpa menggunakan metode *heuristik*.
2. *Software antivirus* yang dibuat hanya bisa berjalan dengan sempurna pada sistem operasi 32-bit dan belum dapat berjalan sempurna pada sistem operasi 64-bit.
3. Dari total 104 sampel *virus/worm* yang peneliti miliki, 82 di antaranya berhasil dideteksi oleh *software antivirus* yang dibuat.

DAFTAR PUSTAKA

[1] Darmal, Achmad. 2009. *War of The Worms Underground Coding*. Jasakom.

[2] Szor, Peter. 2005. *The Art of Computer Virus Research and Defense*. Symantec Press

[3] Nazario, Jose. 2004. *Defense and Detection Strategies against Internet Worms*. Artech House

[4] Sommerville, Ian. 2001. *Software Engineering*. Addison-Weley Publisher

[5] Russel, Stuart J; Norvig, Peter (2003). *Artificial Intelligence : A Modern Approach*. Prentice Hall. New Jersey